**Министерство науки и высшего образования Российской Федерации**
**Федеральное государственное бюджетное образовательное учреждение высшего образования**
**«Московский государственный технический университет имени Н. Э. Баумана**
**(национальный исследовательский университет)»**
**(МГТУ им. Н. Э. Баумана)**

ФАКУЛЬТЕТ «Информатика и системы управления» (ИУ)
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии» (ИУ7)

# Отчет по лабораторной работе №2 по курсу «Анализ алгоритмов»

*«Умножение матриц (сложность)»*

Группа: ИУ7-53Б

Студент: _____ Дьяченко А. А.
(Подпись, дата) (Фамилия И. О.)

Преподаватель: _____ Строганов Д. В.
(Подпись, дата) (Фамилия И. О.)

*Москва, 2023 г.*

# СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

Прямоугольной матрицей называется совокупность чисел, вообще говоря, комплексных, расположенных в виде прямоугольной таблицы, содержащей $n$ строк и $m$ столбцов. [1]

Целью данной лабораторной работы исследование следующих алгоритмов перемножения матриц:

- классического алгоритма;
- алгоритма Винограда;
- оптимизированного алгоритма Винограда;
- алгоритма Штрассена.

Для достижения поставленных целей ставятся задачи:

- реализация требуемых алгоритмов;
- сравнение требуемого времени выполнения реализуемых алгоритмов в тактах процессора и занимаемой памяти;
- подготовка отчёта по лабораторной работе.

# 1 Аналитическая часть

Классический (стандартный) алгоритм перемножения матриц — реализация математического определения умножения матриц. Имеет ассимптотическую сложность $O(n^3)$.

Алгоритм Винограда имеет ассимптотическую сложность $O(n^{2.3755})$, поэтому является одним из самых эффективных по времени алгоритмом умножения матриц [2].

Алгоритм Штрассена имеет ассимптотическую сложность $O(n^{2.78})$ [3]. Идея этого метода состоит в открытии того, что произведение C двух матриц A и B размером 2×2 можно вычислить с помощью только семи, а не восьми умножений, которые необходимы при использовании стандартного алгоритма [4].

# 2 Конструкторская часть

В этом разделе будут представлены схемы реализуемых алгоритмов.

## 2.1 Классический алгоритм перемножения матриц

Схема классического алгоритма перемножения матриц представлена на рисунке 1.

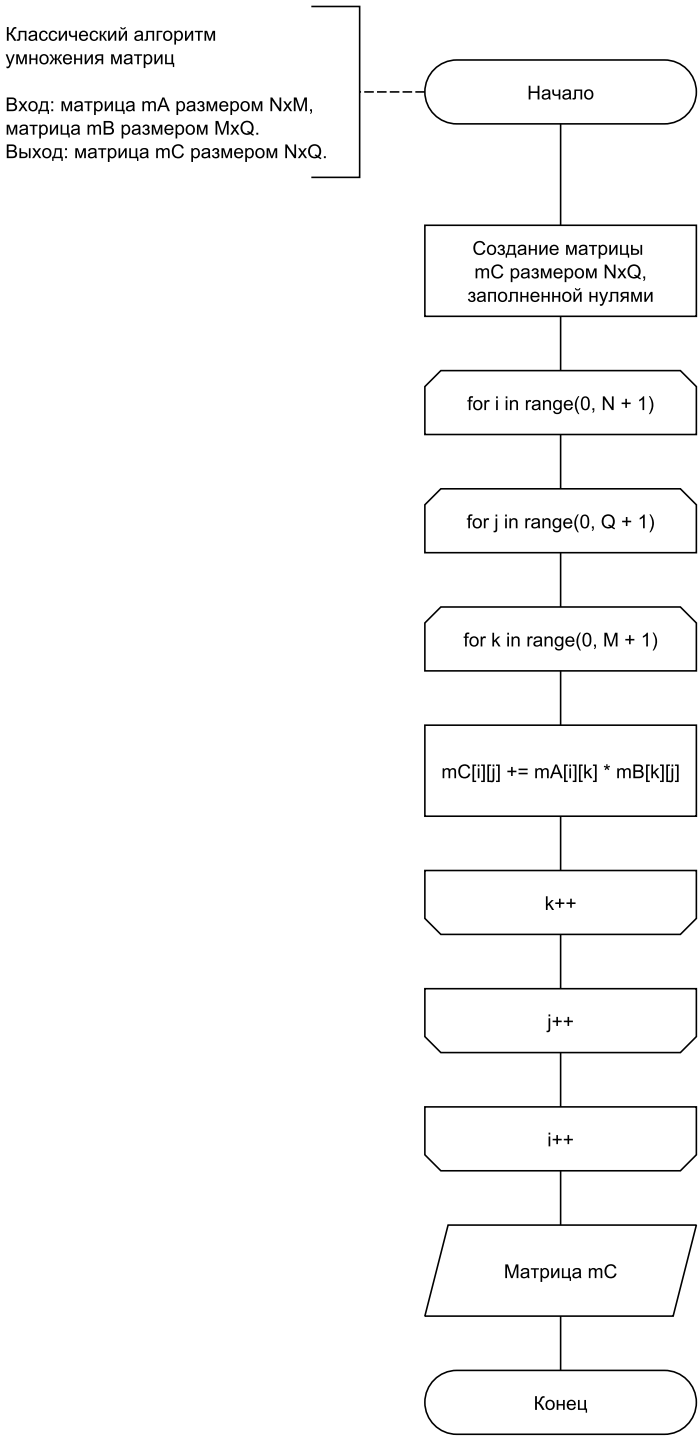


Рисунок 1 – Схема классического алгоритма перемножения матриц

## 2.2 Алгоритм Винограда перемножения матриц

Схема алгоритма Винограда перемножения матриц представлена на рисунке 2 и 3.

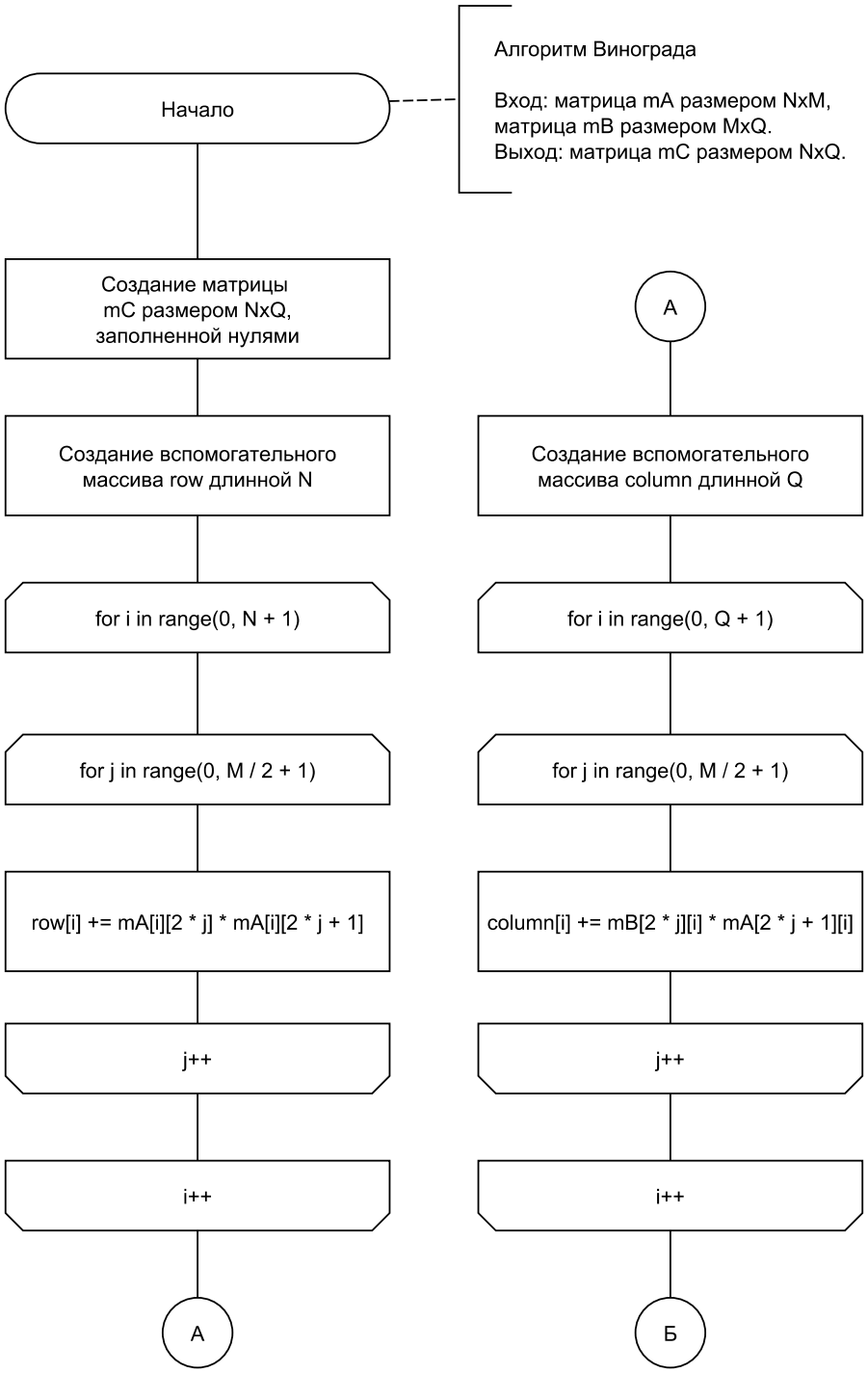


Рисунок 2 – Схема алгоритма Винограда

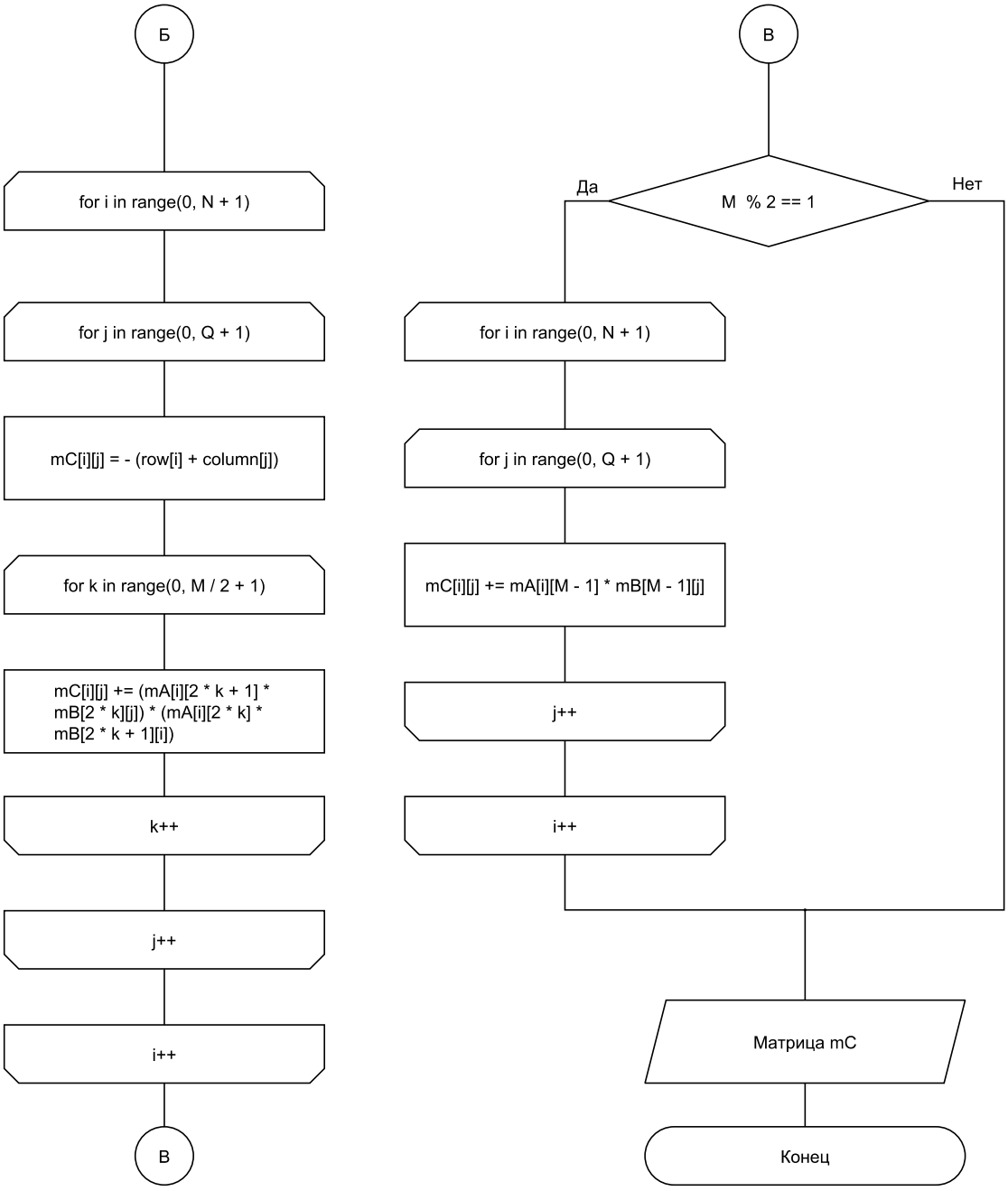


Рисунок 3 – Схема алгоритма Винограда (продолжение)

Оптимизация алгоритма Винограда заключается:

– в замене операций x = x + k на x += k;
– в замене умножения на 2 на побитовый сдвиг;
– в предвычислении некоторых слагаемых.

Для сокращения времени чтения отчёта схема оптимизированного алгоритма не представлена.

## 2.3 Алгоритм Штрассена перемножения матриц

Схема алгоритма Штрассена перемножения матриц представлена на рисунке 4.

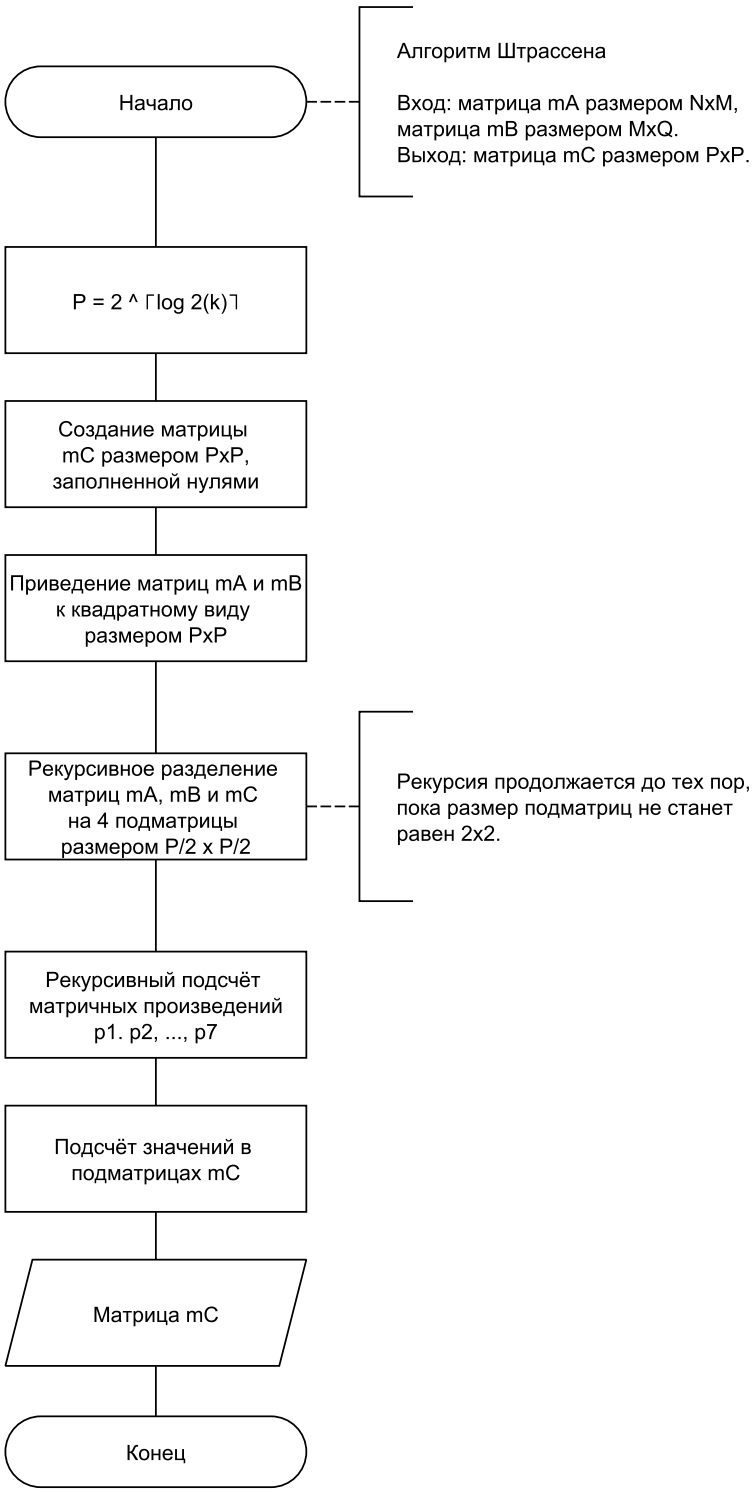


Рисунок 4 – Алгоритм Штрассена

## 2.4 Модель вычислений

Для последующего вычисления трудоемкости необходимо ввести модель вычислений:

а) операции из списка (1) имеют трудоемкость 1;

$$+, -, *, /, \%, ==, !=, <, >, <=, >=, [], ++, -- \tag{1}$$

б) трудоемкость оператора выбора `if условие then A else B` рассчитывается как (2);

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \tag{2}$$

в) трудоемкость цикла рассчитывается как (3);

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}}) \tag{3}$$

г) трудоемкость вызова функции равна 0.

## 2.5 Трудоемкость алгоритмов

В следующих частях будут рассчитаны трудоемкости представленных ранее классического алгоритма и алгоритма Винограда. Трудоёмкость алгоритма Штрассена в отчёте не приведена, поскольку на занятиях не были рассказаны методы рассчёта трудоёмкости рекурсивных функций. Трудоёмкость инициализации результирующей матрицы учитываться не будет, поскольку данное действие есть во всех алгоритмах и не является самым трудоёмким.

Введём обозначения:

- N – кол-во строк первой матрицы;
- M – кол-во столбцов первой матрицы и кол-во строк второй матрицы;
- Q – кол-во столбцов второй матрицы.

### 2.5.1 Классический алгоритм перемножения матриц

Трудоёмкость стандартного алгоритма умножения матриц состоит из:

– внешнего цикла по $i \in [1..M]$, трудоёмкость которого: $f = 2 + M \cdot (2 + f_{body})$;
– цикла по $j \in [1..N]$, трудоёмкость которого: $f = 2 + N \cdot (2 + f_{body})$;
– цикла по $k \in [1..Q]$, трудоёмкость которого: $f = 2 + 10 \cdot Q$.

Трудоёмкость классического алгоритма равна трудоёмкости внешнего цикла. Её можно вычислить, подставив циклы тела (4):

$$f_{classic} = 2 + M \cdot (4 + N \cdot (4 + 10Q)) = 2 + 4M + 4MN + 10MQN \approx 10MQN \quad (4)$$

## 2.6 Алгоритм Винограда

Трудоёмкость алгоритма Винограда состоит из:

– создания и инициализации массивов row и column, трудоёмкость которого (5):

$$f_{init} = N + Q; \quad (5)$$

– заполнения массива row, трудоёмкость которого (6):

$$f_{row} = 2 + N \cdot (2 + \frac{M}{2} \cdot 11); \quad (6)$$

– заполнения массива column, трудоёмкость которого (7):

$$f_{column} = 2 + Q \cdot (2 + \frac{M}{2} \cdot 11); \quad (7)$$

– цикла заполнения для чётных размеров, трудоёмкость которого (8):

$$f_{cycle} = 2 + N \cdot (2 + Q \cdot (23 \cdot \frac{M}{2})); \quad (8)$$

– цикла, для дополнения результирующего массива суммой последних нечётных строки и столбца, если общий размер нечётный, трудоемкость которого (9):

$$f_{last} = \begin{cases} 2, & \text{размер чётный,} \\ 2 + N \cdot (2 + 14Q), & \text{иначе.} \end{cases} \quad (9)$$

Итого, результирующая трудоёмкость алгоритма Винограда равна (10)

$$f_{final} = f_{row} + f_{column} + f_{cycle} + f_{last} \approx 11.5MNQ \tag{10}$$

# 3 Технологическая часть

В данном разделе приведены требования к программному обеспечению, средства реализации и листинги кода.

## 3.1 Требования к ПО

На вход подаются две матрицы. На выходе требуется получить результирующую матрицу – произведение двух введённых.

## 3.2 Средства реализации

В качестве языка программирования для реализации лабораторной работы был выбран C++. Данный выбор обусловлен наличием инструкции [5] для замера количества тактов и поддержкой ООП.

## 3.3 Сведения о модулях программы

Программа состоит из трех программных модулей:

1) main.cpp – главный модуль программы, содержащий меню;
2) algo.cpp, algo.h – модуль с реализацией алгоритмов;
3) io_matrix.cpp, io_matrix.h – модуль ввода/вывода матриц;
4) time.cpp, time.h – модуль замера времени работы алгоритмов.

### 3.4 Реализация алгоритмов

В листинге 1 приведена реализация классического алгоритма умножения двух матриц.

Листинг 1 – Классический алгоритм

```
vector<vector<double>> MatrixSolver::classic_algo() {
    int rowsA = mA.size();
    int colsA = mA[0].size();
    int colsB = mB[0].size();

    vector<vector<double>> res(rowsA, vector<double>(colsB, 0.0));

    for (int i = 0; i < rowsA; i++) {
        for (int j = 0; j < colsB; j++) {
            for (int k = 0; k < colsA; k++) {
                res[i][j] += mA[i][k] * mB[k][j];
            }
        }
    }

    mC = res;
    return res;
}
```

В листинге 2 приведена реализация алгоритма Винограда умножения двух матриц.

Реализация оптимизированного алгоритма Винограда идентична реализации *обычного* алгоритма Винограда, за исключением замены операции умножения на 2 на побитовый сдвиг и предвычисления `half_m = m / 2`.

В этом методе переменная `optimized` отвечает за то, какую реализацию использовать.

$$mode = \begin{cases} \text{алгоритм Винограда}, & \text{optimized} = 0 \\ \text{оптимизированный алгоритм Винограда}, & \text{иначе}. \end{cases} \tag{11}$$

Листинг 2 – Алгоритм Винограда

```
vector<vector<double>> MatrixSolver::winograd_algo(int optimized) {
    int n = mA.size();
    int m = mA[0].size();
    int q = mB[0].size();

    mC.resize(n, vector<double>(q));
    vector<double> row(n, 0);
    vector<double> column(q, 0);

    if (optimized == 1) {
        int half_m = m >> 1;
        for (int i = 0; i < n; i++)
            for (int j = 0; j < half_m; j++)
                row[i] += mA[i][j << 1] * mA[i][(j << 1) + 1];

        for (int i = 0; i < q; i++)
            for (int j = 0; j < half_m; j++)
                column[i] += mB[j << 1][i] * mB[(j << 1) + 1][i];

        for (int i = 0; i < n; i++)
            for (int j = 0; j < q; j++) {
                mC[i][j] = -row[i] - column[j];
                for (int k = 0; k < half_m; k++)
                    mC[i][j] += (mA[i][k << 1] + mB[(k << 1) + 1][j
                        ]) * (mA[i][(k << 1) + 1] + mB[k << 1][j]);
            }
    }
```

Листинг 3 – продолжение листинга 2

```
    else {
        for (int i = 0; i < n; i++)
            for (int j = 0; j < m / 2; j++)
                row[i] += mA[i][2 * j] * mA[i][2 * j + 1];

        for (int i = 0; i < q; i++)
            for (int j = 0; j < m / 2; j++)
                column[i] += mB[2 * j][i] * mB[2 * j + 1][i];

        for (int i = 0; i < n; i++)
            for (int j = 0; j < q; j++) {
                mC[i][j] = -row[i] - column[j];
                for (int k = 0; k < m / 2; k++)
                    mC[i][j] += (mA[i][2 * k] + mB[2 * k + 1][j]) *
                        (mA[i][2 * k + 1] + mB[2 * k][j]);
            }
    }

    if (m % 2 == 1)
        for (int i = 0; i < n; i++)
            for (int j = 0; j < q; j++)
                mC[i][j] += mA[i][m - 1] * mB[m - 1][j];
    return mC;
}
```

В листинге 4 приведена реализация алгоритма Штрассена умножения двух матриц.

Листинг 4 – Алгоритм Штрассена

```
#define strassen strassenen_algo
#define m_sub matrix_sub
#define m_add matrix_add

vector<vector<double>> MatrixSolver::strassen(const vector<vector<
    double>>& A, const vector<vector<double>>& B) {
    int n = A.size();

    if (n == 1) {
        vector<vector<double>> C(1, vector<double>(1, 0));
        C[0][0] = A[0][0] * B[0][0];
        return C;
    }

    int half_n = n / 2;
    vector<vector<double>> A11(half_n, vector<double>(half_n));
    vector<vector<double>> A12(half_n, vector<double>(half_n));
    vector<vector<double>> A21(half_n, vector<double>(half_n));
    vector<vector<double>> A22(half_n, vector<double>(half_n));
    vector<vector<double>> B11(half_n, vector<double>(half_n));
    vector<vector<double>> B12(half_n, vector<double>(half_n));
    vector<vector<double>> B21(half_n, vector<double>(half_n));
    vector<vector<double>> B22(half_n, vector<double>(half_n));

    for (int i = 0; i < half_n; i++) {
        for (int j = 0; j < half_n; j++) {
            A11[i][j] = A[i][j];
            A12[i][j] = A[i][j + half_n];
            A21[i][j] = A[i + half_n][j];
            A22[i][j] = A[i + half_n][j + half_n];
            B11[i][j] = B[i][j];
            B12[i][j] = B[i][j + half_n];
            B21[i][j] = B[i + half_n][j];
            B22[i][j] = B[i + half_n][j + half_n];
        }
    }
```

Листинг 5 – продолжение листинга 4

```
vector<vector<double>> P1 = strassen(A11, m_sub(B12, B22));
vector<vector<double>> P2 = strassen(m_add(A11, A12), B22);
vector<vector<double>> P3 = strassen(m_add(A21, A22), B11);
vector<vector<double>> P4 = strassen(A22, m_sub(B21, B11));
vector<vector<double>> P5 = strassen(m_add(A11, A22), m_add(B11,
    B22));
vector<vector<double>> P6 = strassen(m_sub(A12, A22), m_add(B21,
    B22));
vector<vector<double>> P7 = strassen(m_sub(A11, A21), m_add(B11,
    B12));

vector<vector<double>> C11 = matrix_add(matrix_sub(matrix_add(P5,
    P4), P2), P6);
vector<vector<double>> C12 = matrix_add(P1, P2);
vector<vector<double>> C21 = matrix_add(P3, P4);
vector<vector<double>> C22 = matrix_sub(matrix_sub(matrix_add(P5,
    P1), P3), P7);

vector<vector<double>> C(n, vector<double>(n, 0));
for (int i = 0; i < half_n; i++) {
    for (int j = 0; j < half_n; j++) {
        C[i][j] = C11[i][j];
        C[i][j + half_n] = C12[i][j];
        C[i + half_n][j] = C21[i][j];
        C[i + half_n][j + half_n] = C22[i][j];
    }
}

mC = C;
return C;
}
```

# 4 Исследовательская часть

## 4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялся замерный эксперимент:

— операционная система Windows 11;

— память 16 ГБ;

— процессор 3,6 ГГц 6-ядерный процессор AMD Ryzen 5000 series 5.

Замеры проводились на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только интегрированной средой разработки и непосредственно выполняемой программой.

## 4.2 Пример работы программы

На рисунке 5 представлен пример работы программы. Пользователь выбирает алгоритм умножения двух матриц из списка меню. Вводит размеры матриц, их значения. Программа вычисляет результат умножения и выводит его на экран.

```
        Меню

1. Классический алгоритм
2. Алгоритм Винограда
3. Оптимизированный алгоритм Винограда
4. Алгоритм Штрассера
5. Замеры времени
0. Выход

Выбор: 1
Введите:
Кол-во строк первой матрицы A: 2
Кол-во столбцов первой матрицы A: 2
Кол-во столбцов второй матрицы Б: 3
Введите элементы матрицы A:
1 2 3 4
Введите элементы матрицы B:
1 2 3 4 5 6
Матрица A:
1 2
3 4
Матрица B:
1 2 3
4 5 6
Матрица C:
9 12 15
19 26 33
```

Рисунок 5 – Пример работы программы

### 4.3 Время выполнения реализованных алгоритмов

Замеры времени работы реализованных алгоритмов для определенного размера квадратных матриц проводились 1000 раз, при этом каждый раз значения матриц генерировались случайно.

Для измерения тактового времени была использована инструкция rdtsc [5].

В качестве результата, представленного на графике 6, взято среднее время выполнения алгоритмов в тактах процессора для каждой матрицы размера от 1 до 10.

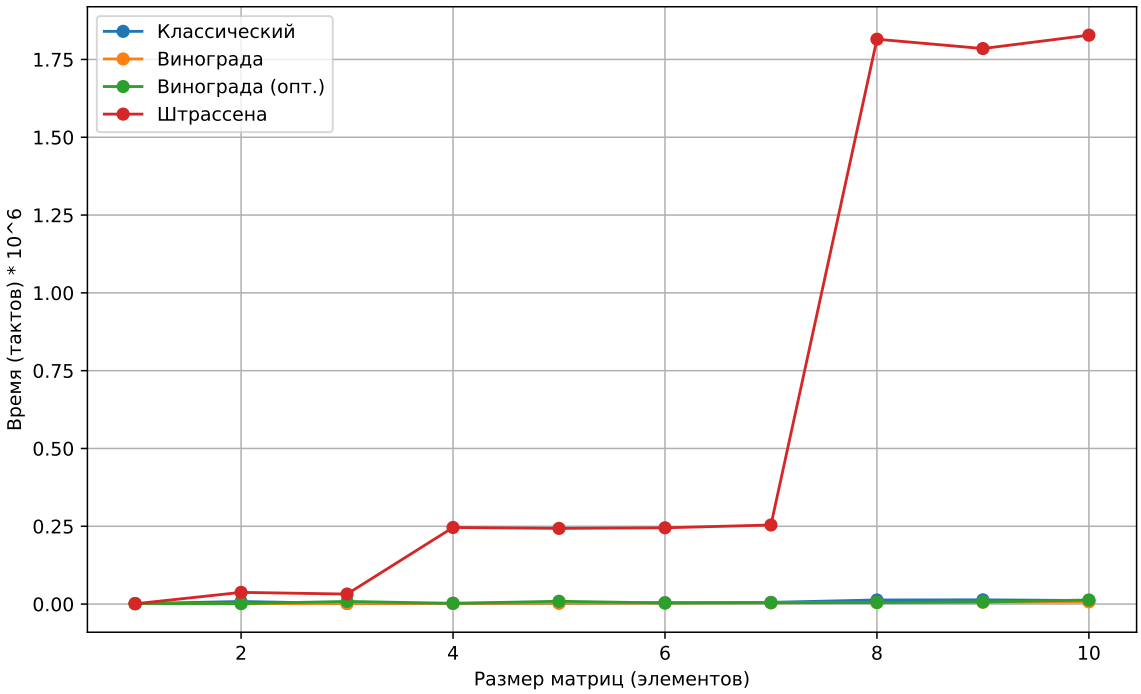


Рисунок 6 – Время выполнения алгоритмов

Тот же самый график, но без учёта алгоритма Штрассена:

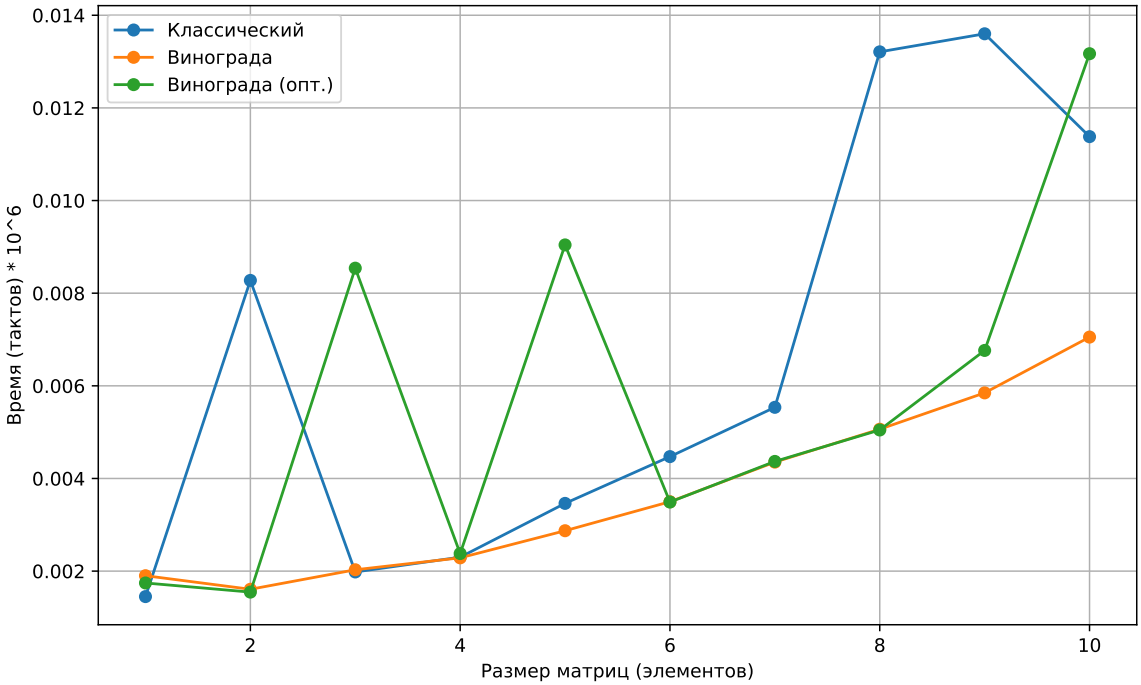


Рисунок 7 – Время выполнения алгоритмов (без учёта алгоритма Штрассена)

В половине случаев оптимизированная версия алгоритма Винограда работает дольше, чем *обычный* алгоритм Винограда.

В результате трансляции кода программы на язык ассемблера, было замечено следующее: дизассемблированная строка исходного кода, выполняющая операцию вычисления значения `row[i]`, при оптимизированной реализации алгоритма Винограда содержит инструкции `lea` и `movsxd`, из-за чего количество выполняемых команд становится больше, чем в *обычном* алгоритме Винограда.

Количество команд и, следовательно, разность в количестве инструкций также увеличивается при нечётном размере матриц. Поэтому на графике заметны выбросы во времени у оптимизированного алгоритма Винограда.

Поэтому время выполнения увеличивается.

## 4.4 Занимаемая память реализованных алгоритмов

График 8 демонстрирует объём памяти в байтах, потребляемый разными реализациями алгоритмов в зависимости от размера матриц. Занимаемый объём памяти считался как сумма размеров переменных, используемых в алгоритме и структуре данных.

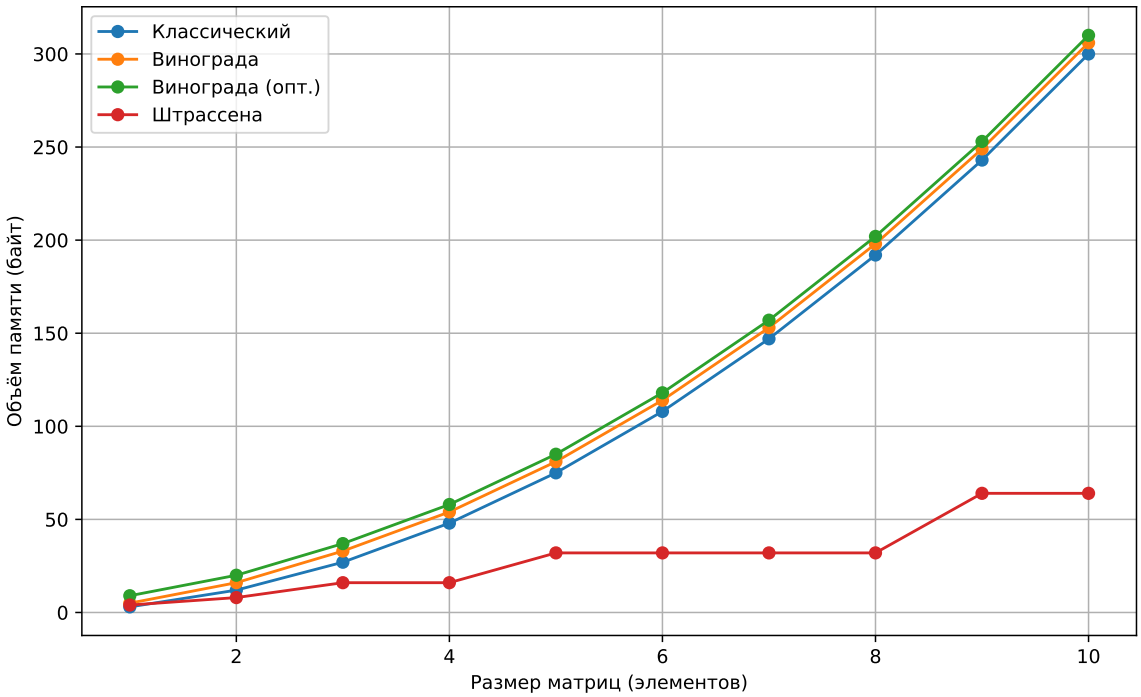


Рисунок 8 – Объём памяти, используемый алгоритмами

## 4.5 Вывод

В результате анализа замеров времени выполнения и затрат памяти на различных алгоритмах были сделаны следующие выводы:

- алгоритм Штрассена занимает меньше всего памяти при размере квадратных матриц больше единицы;
- алгоритм Винограда и его оптимизированная версия быстрее всех умножают матрицы;
- разница во времени выполнения алгоритма Винограда и его оптимизированной версии зависит от процессора.

## ЗАКЛЮЧЕНИЕ

Цель работы достигнута: исследованы следующие алгоритмы перемножения матриц:

- классический алгоритм;
- алгоритм Винограда;
- оптимизированный алгоритм Винограда;
- алгоритм Штрассена.

В ходе выполнения лабораторной работы были решены все задачи:

- реализованы требуемые алгоритмы;
- проведено сравнение требуемого времени выполнения реализуемых алгоритмов в тактах процессора и занимаемой памяти;
- подготовлен отчёт по лабораторной работе.

Замеры показали, что алгоритм Винограда и его оптимизированная версия являются самыми эффективными *по времени* начиная с длины квадратных матриц, равной четырём. Классический алгоритм эффективнее других *по времени* только для случаев перемножения матриц длинной 1 и 3. Алгоритм Штрассена является самым эффективным *по памяти*, начиная с длины матриц, равной двум.

Рассчитанная трудоёмкость классического алгоритма перемножения матриц меньше, чем трудоёмкость алгоритма Винограда. Но, несмотря на это, начиная с длины квадратных матриц, равной четырём, время выполнения реализации классического алгоритма становится больше, чем время выполнения реализации алгоритма Винограда.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


1. *К. Ф. Д., Н. Ф. В.* Вычислительные методы линейной алгебры. — 2009. — С. 7.
2. *Анисимов Н. С., Строганов Ю. В.* Реализация алгоритма умножения матриц по Винограду на языке Haskell // Новые информационные технологии в автоматизированных системах. — 2018. — С. 390—395. — URL: https://cyberleninka.ru/article/n/realizatsiya-algoritma-umnozheniya-matrits-po-vinogradu-na-yazyke-haskell.
3. *B. K.* Ultra-Fast Matrix Multiplication: An Empirical Analysis of Highly Optimized Vector Algorithms. — 2004. — URL: https://cs.stanford.edu/people/boyko/pubs/MatrixMult_SURJ_2004.pdf.
4. *Лапина Н. Р., Булатникова М. Е.* АЛГОРИТМ ШТРАССЕНА ДЛЯ УМНОЖЕНИЯ МАТРИЦ // МАТЕМАТИКА И ЕЕ ПРИЛОЖЕНИЯ В СОВРЕМЕННОЙ НАУКЕ И ПРАКТИКЕ. — 2014. — С. 41—45. — URL: https://www.elibrary.ru/item.asp?id=23140890.
5. *Microsoft.* RDTSC (Read Time-Stamp Counter). — URL: https://learn.microsoft.com/ru-ru/cpp/intrinsics/rdtsc?view=msvc-170.